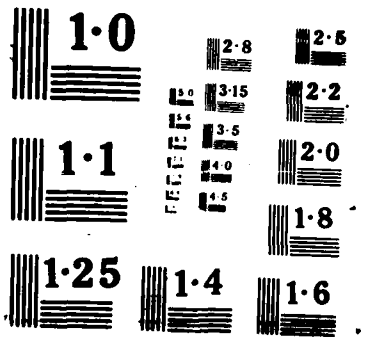
AD-A176 701   A PROPOSAL FOR A PROFILE INTERCHANGE FORMAT PART 1   1/1
SYNTAX PART 2 SEMANTIC..(U) MASSACHUSETTS INST OF TECH
CAMBRIDGE LAB FOR COMPUTER SCIENCE.. D S BONING ET AL.
UNCLASSIFIED   DEC 86 VLSI-MEMO-86-356 N00014-80-C-0622   F/G 9/2   NL


1·0
2·8
2·5
3·15
2·2
3·5
1·1
2·0
4·0
4·5
1·8
1·25
1·4
1·6

12

MASSACHUSETTS INSTITUTE OF TECHNOLOGY VLSI PUBLICATIONS

VLSI Memo No. 86-356
December 1986 Contract N00014-80-C-0622

A PROPOSAL FOR A PROFILE INTERCHANGE FORMAT
PART I: SYNTAX
PART II: SEMANTICS

Duane S. Boning and Thye-Lai Tung

Abstract

This paper examines issues in the design and implementation of a Profile Interchange Format (PIF) for use in the standard representation of semiconductor profile and attribute information. A distinction is made between the syntax and the semantics of the representation; both facets of the PIF are discussed. Note that this is a preliminary working document, and as such represents only approximately constantly evolving thoughts and implementations.

~~In~~ Part I, syntactic issues of the PIF are ~~examined~~. ~~Our~~ decision has been to develop two versions of the format: an ASCII format (for exchange of information), and a BINARY format (for local use by individual CAD tools). For the ASCII form, ~~we have chosen~~ a highly restricted LISP-like syntax to capture hierarchy and to ease parsing of the structure. A binary version of the format (~~which~~ we call ~~l~~SNC) has been prototyped, and allows highly efficient storage and access of PIF information. While the binary and ASCII syntactic mechanisms have been implemented, work remains to define a standard semantics (an organization and minimum set of names and data) for the exchange of process and device simulation information.

Part II examines issues in the semantic representation of semiconductor profile and device structure information, focusing on the actual organization of device and profile data. Topics examined include general geometry representation, association of attribute information with arbitrary parts of the geometry, a definition-reference mechanism, and transformation specification. Finally, additional syntax issues in the specification of a PIF standard are discussed.

Microsystems Research Center Room 39-321
Massachusetts Institute of Technology
Cambridge Massachusetts 02139
Telephone (617) 253-8138

A Proposal for a Profile Interchange Format
Part I: Syntax
Part II: Semantics

by

Duane S. Boning[1] and Thye-Lai Tung[2]

Massachusetts Institute of Technology
Room 39-315, Cambridge, MA 02139

June 4, 1986

Abstract

This paper examines issues in the design and implementation of a Profile Interchange Format (PIF) for use in the standard representation of semiconductor profile and attribute information. A distinction is made between the *syntax* and the *semantics* of the representation; both facets of the PIF are discussed. Note that this is a preliminary working document, and as such represents only approximately constantly evolving thoughts and implementations.

In Part I, syntactic issues of the PIF are examined. Our decision has been to develop two versions of the format; an ASCII format (for exchange of information), and a BINARY format (for local use by individual CAD tools). For the ASCII form, we have chosen a highly restricted LISP-like syntax to capture hierarchy and to ease parsing of the structure. A binary version of the format (which we call SNC) has been prototyped, and allows highly efficient storage and access of PIF information. While the binary and ASCII *syntactic* mechanisms have been implemented, work remains to define a standard *semantics* (an organization and minimum set of names and data) for the exchange of process and device simulation information.

Part II examines issues in the semantic representation of semiconductor profile and device structure information, focussing on the actual organization of device and profile data. Topics examined include general geometry representation, association of attribute information with arbitrary parts of the geometry, a definition-reference mechanism, and tranformation specification. Finally, additional syntax issues in the specification of a PIF standard are discussed.

[1] Network addres boning@caf.mit.edu
[2] Network address tung@caf.mit.edu

A Proposal for a Profile Interchange Format
Part I: Syntax

by

Duane S. Boning[1] and Thye-Lai Tung[2]

Massachusetts Institute of Technology
Room 39-315, Cambridge, MA 02139

June 4, 1986

Abstract

A proposed syntax for the Profile Interchange Format (PIF) is presented, suitable for use as a standard representation of semiconductor profile and device structure and attribute information. A distinction is made between the actual *syntax* and the *semantics* of the representation. For the ASCII form of the format, we have chosen a highly restricted LISP-like syntax to capture hierarchy and to ease parsing of the structure. A binary form of the format (which we call SNC) has been prototyped, and allows highly efficient storage and access of PIF information. While the binary and ASCII *syntactic* mechanisms have been implemented, work remains to define a standard *semantics* (an organization and minimum set of names and data) for the exchange of process and device simulation information.

[1]Network addres boning@caf.mit.edu
[2]Network address tung@caf.mit.edu

1 Introduction

A number of groups involved in the development or integration of process and device simulation programs have come to recognize the need for a profile interchange format [1], [2]. The intent of this document is to report ongoing work at MIT in the development of such a format, called here PIF. The philosophy, guidelines, and goals of the PIF will be examined. A discussion of the distinction between the format "syntax" and format "semanctics" is made, and the need for both a textual and a binary expression of the format is pointed out. We then discuss the binary version of the format in some detail, and present examples of comparable ASCII formats. Finally, we close with some comments about the adoption of a standard built on this or another format.

2 Philosophy, Guidelines, and Goals

Before we present the proposed Profile Interchange Format, we need to consider the problem under attack. This section examines, first of all, the potential uses of a profile interchange format. This will be followed by a discussion of the requirements and constraints on an interchange format.

2.1 Uses of a PIF

In previous meetings of the informal profile interchange format standards subcommittee, there has been some confusion regarding the magnitude of the work involved in designing an interchange format. This confusion was due to the various groups of workers envisioning different uses for the format. In order to be successful, development of a working standard must proceed with all of these potential uses in mind. Development of a standard that will meet these multiple needs will not be trivial.

There are two major categories of uses for the PIF. First, the interchange format can be used to transport structure and attribute information from one site to another. Secondly, some version of the format will be used locally to represent and store information in a "database" mode. The two uses are not at odds with each other; the need to represent completely structure and attribute information is common to both. An argument can be made that "information transfer" is simply a special case of the "information capture" problem. That is, once we have a general format we can use it either as a database (almost certainly in a binary form) or as a file transfer format (almost certainly in an ASCII form).

If we accept that the format may appear in two modes, we still must ask to what *use* the format will actually be put. The following is a list of possible uses of a profile interchange format:

- As a means for representing *process simulation output* (both structure or geometry and attribute information).

- As a standard way of representing *profile structures* (and attributes of that structure), whether these have been produced by simulators, by test or measurement equipement, or directly by the user.

- As a means for representing *device simulation input structures*. This may be a different (more comprehensive) representation than that needed for process simulation output.

- As a means for representing *device simulation results*.

2.2 PIF Representational Requirements

The above possible uses of a profile interchange format place certain demands on the PIF. The PIF must be powerful enough to handle several classes of information; the representational demands on the PIF are enumerated below.

- The PIF must allow a hierarchy of geometric objects to be specified. This hierarchy must include one-dimensional objects (points, grids, boundaries, and regions), two-dimensional objects (points, segments, elements, various grids, boundaries, and regions), and even three-dimensional objects (with surfaces and three-dimensional meshes as well as two dimensional objects). The representation must be able to capture the hierarchical geometry inherent in real and simulated structures.

- The PIF should allow attributes to be defined on any part of that geometry. These attributes might be scalars, vectors, arrays, or strings, and may be associated with any part of the geometry described above.

- A hierarchy of data is needed as well. Thus, one piece of data may be an attribute of another piece of data rather than simply an attribute of a part of the geometry.

- Lastly, there may be a need to associate various geometries or parts of a geometry with a piece of data. This is different than the first need above, where a geometry is considered to be composed only of other smaller geometries. Here, a structure may depend on (and therefore need to be associated with) some data value. For instance, the geometry in a simulation often depends upon simulation time; the interchange format should be capable of representing this association explicitly.

2.3 Constraints on the PIF

There are a number of indirect demands on the interchange format. These are imposed in order that the format be accessible to the largest number of applications.

- The format (or some well defined subset of the format) should be simple enough that application programs can read or write the format directly.

- The format should require a minimum of "interpretation" of information. The format must be able to capture the information, but should not have to know what that information "means."

- The format must be able to coexist with both ASCII and binary formats of data. The amount of information required and generated by simulation, particularly two-dimensional programs, is so large that efficiency in storage and access speed is an important consideration.

- The format should be compatible in concept with existing standard format work, particularly EDIF [3].

- The format should be implementable in standard languages, particularly C and FORTRAN.

3 PIF and SNC

3.1 Syntax vs Semantics

We have found it to be useful to distinguish between the *syntax* and the *semantics* of the format. The problem of developing an appropriate syntax is to find a reasonable form for storage of data. In our implementation, the *syntax* really refers to the general appearance of data in an ASCII file. This same data might be stored in purely binary form using SNC; while the "appearance" of the data is quite different, the conceptual structure of the binary file maps directly to the ASCII syntax. The *semantics*, on the other hand, has to do with the actual profile or device information expressed using the available syntax. Development of a *standard*, then, requires establishing agreed upon conventions of the kinds of information simulation programs will read and write, as well as agreeing upon the syntax of the format itself.

It is useful to decouple the two concepts; as the format evolves, the distinction becomes more important. General software that can read and write information using the syntax is the primary programming task. If the syntax itself does not change, and if the semantics of the format are not coded directly into the syntax, then the semantics can change without the basic software becoming obsolete. That is, the exact names and type of information in the "standard" can evolve over time, while the format syntax and tools for reading that format remain unchanged.

3.2 ASCII vs. binary

A second important distinction is between the ASCII and binary forms of storage. In our implementation, the ASCII form is intended for two purposes. First, the ASCII form is human readable, and can be generated by hand if need be. Secondly, the ASCII form is suitable for shipment of information from one site to another. Because this ASCII format serves as the link between different sites, a *standard* syntax and semantic base is required.

The binary form of the Profile Interchange Format has been optimized for compactness in the storage of information and for speed in accessing that information. In our view, the binary form is the only reasonable solution for day to day use of an interchange format. There are two primary obstacles to a binary format which must be overcome in any standards effort. The first problem in that a binary format depends on computer architecture, so that the actual binary files can not themselves be uniform across all sites. The second problem is that a binary form is not directly human readable. We believe there are reasonable solutions to these two problems.

The exact nature of the binary format need not be uniform across all sites. Since many sites will typically be using the same application programs, it is only necessary that there be a uniform, *standard* software interface between applications and the binary format. That is, a *standard* set or package of routine calls for reading and writing the format (whether in its ASCII or binary form) is needed; how the information then gets stored in a local binary file can be site dependent. Our package or library of read and write routines (which is what SNC really is) is one example of such an interface.

The distinctions between syntax and semantics, as well as between the binary and ASCII form should become more clear once the particulars are examined.

4 ASCII version of PIF

This section examines the syntactic structure of the ASCII form of the Profile Interchange Format. No suggestions or recommendations are made here for a standard set of names or data structures, only for a general syntax for textual storage of information.

The basic syntax chosen bears resemblance to LISP. The drawback of a LISP-like structure, or of any sequential data storage technique, is that directory or name information is interspersed with the bulk of the data. Thus, retrieval of a specific datum requires interpretation of the file itself. This drawback is the principle driving force behind the development of SNC, a binary, random access storage form described in a later section.

The basic grouping of information is an *entry*, consisting of a name, optional datatype information, and optional data, as illustrated in Figure 1.

$$\begin{gathered}(name\ [data\text{-}type\text{-}information]\ [data] \\ [(additional\text{-}entries)]\ )\end{gathered}$$

Figure 1: Basic PIF syntax.

The syntax we have chosen is a heavily-restricted form of the LISP syntax, and really comes to bear only superficial resemblances to LISP. The first word after each '(' must be the name of the data item. The second field may be either another *entry*, or it may be a *type specifier* such as %string or %int. Some type specifiers require additional arguments, such as the number of array dimensions and the size of the array in each dimension; these arguments follow the type specifier (separated by white space). Finally, the data itself appears. A few short examples are shown in Figure 2.

```
(date %string "March 4, 1952")
(time %string 12:00)
(surface-states
        (fast-states %real 1.5e10)
        (slow-states %real 1.2e9))
(arsenic-conc %realarray 1 5
        1.5e20 1.51e20 1.5325e20 1.555e20 1.6e20)
```

Figure 2: ASCII Format examples.

5 SNC

A prototype version of a package to read and write a binary version of the PIF has been implemented [4]. The SNC package, consisting of a library of C routines, presents a common interface to Fortran and C application programs wishing to write or read information in binary SNC files. The actual file structure of SNC is detailed below. A brief description of the routines making up the interface is presented, and examples of SNC use are shown.

5.1 SNC File Structure

In the prototype implementation of SNC, all directory information and data is maintained within an SNC binary file as illustrated in Figure 3. Each group of data has a directory entry associated with it, as shown in Figure 4, containing information on the size of the directory entry, the name of the entry, the level in the hierarchy of the data, and the location of the data within the file. These directory entries are sequential with respect to the hierarchy of the information stored in the file, and are located physically near to each other in the file.

The directory entries may be scanned efficiently either to determine the contents of the file, or to locate a particular named entry within the file. This is a marked improvement

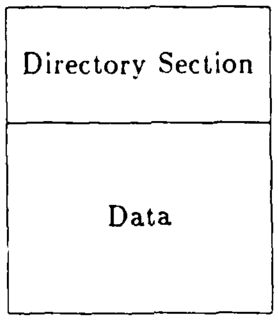


Figure 3: SNC binary file structure.

entry size	byte
data type	byte
level	byte
name length	byte
name	byte[name length]
base unit	byte
number dims	byte
dimensions	long[number dims]
pointer	long
entry size	byte

where a "long" is 32 bits and a "byte" is 8.

Figure 4: SNC directory entry.

over the use of "headers" at the start of each group of data, since the data itself does not need to be scanned in order to locate the data.

5.2 Write Interface — C

One goal of SNC is to allow an application to output compactly and naturally the information it wishes. The binary format does not restrict or dictate the actual data structures used by application programs. Data structures are already in place in existing programs and would be difficult to modify; furthermore, each application must be allowed to tailor its own internal data structures to the problem at hand. The intent of the SNC and the PIF in general is to allow an application program to write or read, in a uniform manner, information for interchange with other programs. Care has been taken in the design of SNC to make it possible to mimic very closely typical data structures in conventional high level languages such as C, Pascal, and PL/I. The following routines make up the SNC write interface for application programs written in C:

```
Routines for opening and closing the SNC file:
    snc_write_open(filename)
        char *filename;
    snc_write_close( )


Hierarchy creation routines:
    snc_make_level(name)
        char *name;
    snc_enter_level( )
    snc_exit_level( )


Basic data writing routines:
    snc_write_flag(name)
        char *name;
    snc_write_string(name, stringval)
        char *name, *stringval;
    snc_write_real(name, realval)
        char *name;
        float realval;
    snc_write_real_vector(name, size, valuep)
        char *name;
        int size;
        float valuep[ ];
    snc_write_real_array(name, num_actual_dims, actual_dims, valuep)
        char *name;
        int num_actual_dims;
        int actual_dims[num_actual_dims];
        float valuep[ ];
    snc_write_int(name, intval)
        char *name;
        int intval;
```

```
Generalized write of array information (element at a time)
    snc_write_start_real(name, num_write_dims, write_dims)
        char *name;
        int num_write_dims;
        int write_dims[num_write_dims];
    snc_write_real_item(value);
        float value;
    snc_write_end_real( );
```

5.3 Read Interface — C

The following routines make up the SNC read interface for application programs written in C:

```
Routines for opening and closing the SNC file:
    snc_read_open(filename)
        char *filename;
    snc_read_close( );
```

```
Directory searchand information routines:
    snc_locate_entry(name)
        char name[ ];
    snc_get_directory(name)
        char name[ ];
    entry *snc_get_entry(name)
        char *name;
        where entry = struct {
            int level, type, size, num_dims;
            int dims[ ];
            }
    snc_enter_level( );
    snc_exit_level( );
```

```
Basic data reading routines:
    snc_read_string(valuep)
        char valuep[ ];
    snc_read_int(valuep)
        int *valuep;
    snc_read_real(valuep)
        float *valuep;
    snc_read_real_vector(valuep)
        float valuep[ ];
```

```
Generalized read of array information (element at a time):
    snc_read_real_start(num_actual_dims,actual_dims)
        int num_actual_dims,
        int actual_dims[num_actual_dims];
    snc_read_real_item(valuep,dim1,dim2,dim3,dim4)
        float *valuep;
        int dim1,dim2,dim3,dim4;
```

5.4 Examples

The following examples illustrate the use of these routines from inside an application program. Conceptually, the application programmer may work as though he were writing out the ASCII version of the format. His concerns are with the hierarchical presentation of his data, and in the correct and appropriate naming of that data. The data is put into a binary form by SNC automatically, and the application is shielded from the particulars of the storage.

WRITE EXAMPLE

```
#include "sncdef.h"
#include "sncerror.h"

static float grid[5] = {0.0,1.0,2.0,3.0,4.0};
static float boron[5] = {1.0e20 1.1e20 1.2e20 1.3e20 1.4e20};
static float arsenic[5] = {5.0e20 5.1e20 5.2e20 5.3e20 5.4e20};

main ()
  { float actE = 10.0;
    char *run_name = "test-run";

    snc_write_open("testfile.snc");
    snc_make_level(run_name);
    snc_write_string("Date","April 29, 1986");
    snc_write_real_vector("Grid_Spacing",5,data);
    snc_makelevel("Boron");
    snc_write_real_vector("concentration", 5, data);
    snc_write_real("activation-enery", &actE);
    snc_makelevel("characteristics");
    snc_write_string("plot-color","RED");
    snc_exit_level();
    snc_exit_level();
    snc_makelevel("Arsenic");
    snc_write_real_vector("concentration", 5, data);
    snc_exit_level();
    snc_write_int("Bye", 4);
    snc_close();
  }
```

The resulting SNC file can be either translated to an ASCII file which can be read in conventional ways, or it can be examined with "decoding" utilities directly. In either case, the corresponding ASCII format to the above example would be:

```
(test-run
    (Date %string “April 29, 1986”)
    (Grid_Spacing %realarray 1 5
              0.0 1.0 2.0 3.0 4.0)
        (Boron
              (concentration %realarray 1 5
                   1.0e20 1.1e20 1.2e20 1.3e20 1.4e20)
              (activation-energy %real 10.0)
              (characteristics
                   (plot-color %string RED)))
        (Arsenic
              (concentration %realarray 1 5
                   5.0e20 5.1e20 5.2e20 5.3e20 5.4e20))
        (Bye %int 4))
```

READ EXAMPLE

```
#include <stdio.h>
#include "sncdef.h"
#include "sncerror.h"

main ()
  { char name[100];
    float conc[100];
    float grid[100];
    ENTRY_STRUCT *entry;
    int i;

    snc_open_read("testfile.snc");

    snc_locate_entry("test-run");
    snc_enter_level();
    snc_locate_entry("Grid-spacing");
    snc_read_real_vector(grid);
    snc_locate_entry("arsenic");
    snc_enter_level();
    entry = snc_get_entry("concentration");
    snc_read_real_vector(conc);
    snc_close_read();
    printf("Arsenic concentration for test was:\n");
    for (i=0; i<entry->size; i++) printf("\t %f  %f\n",grid[i],conc[i]);
  }
```

6 Recommendations for a Standard

Development of a standard profile interchange format requires agreement in two areas:

- Choice of syntax.
- Choice of semantics. A minimal set of names and structures to be used for process structures, for device structures, and for device simulation output is needed.

The bulk of this document has described a proposal for a syntax (both ascii and binary) appropriate for device and profile information storage. Little has been mentioned about exactly what information should be stored, and what conventions should be followed in expressing that information using the available syntax. While we have been considering possible choices, work on this part of the interchange format is in a much more formative state (as of this writing, at least).

The current status of work at MIT is as follows.

- A prototype implementation of the SNC package has been completed (written in C).
- Preliminary C and Fortran interface routines have been defined.
- Utilities to convert from the ASCII to the SNC form, and from SNC to the ASCII are currently being implemented.
- An interactive utility to read SNC files and present textual and graphical data is under development.

- A utility to aid the application programmer in using SNC is under consideration. In particular, this utility would perform post-mortem analysis of faulty files, and aid the programmer in debugging his use of SNC.
- Consideration of various conventions for expressing typical process and device simulation structures is underway. We consider SNC to be a reasonable solution to the problem of a profile interchange format syntax. We are now looking at the problem of PIF semantics as well.

7 Acknowledgements

These thoughts have developed out of meetings with parties interested in using a profile interchange format (the informal U.S. subcommittee under the direction of Andy Neureuther and Bob Dutton), as well as with colleagues here at MIT. This latter group includes Duane Boning, Thye-Lai Tung, Dimitri Antoniadis, Jarvis Jacobs, Robert Harris, Ron Duncan, and Ralph Iverson.

References

[1] A. R. Neureuther, "Profile Interchange Format," 1985. Working notes.

[2] S. Duvall and D. Lucey, "An Interchange Format for Process and Device Simulation," Jan. 1986. Personal communication.

[3] *EDIF Specification — Version 1 1 0.* Electronic Design Interchange Format Steering Committee, Nov. 1985.

[4] T. Tung, "SNC: an interchange format for simulation programs," July 1985. Personal communications and accompanying software.

A Proposal for a Profile Interchange Format
Part II: Semantics

by

Duane S. Boning[1] and Thye-Lai Tung[2]

Massachusetts Institute of Technology
Room 39-315, Cambridge, MA 02139

November 6, 1986

Abstract

This paper examines issues in the semantic representation of semiconductor profile and device structure information. An earlier paper discussed the need for both an ASCII *interchange* format and a BINARY *database* type format for local use by the individual tools. Prototype proposals for both the ascii syntax (PIF) and the binary syntax (SNC) were discussed. This paper, on the other hand, focusses on the actual organization of device and profile data. Topics examined include general geometry representation, association of attribute information with arbitrary parts of the geometry, a definition-reference mechanism, and tranformation specification. Finally, additional syntax issues in the specification of a PIF standard are discussed.
NOTE: This is a very preliminary working document, and as such is in a very incomplete (and probably unintelligible) form.

[1] Network addres boning@caf.mit.edu
[2] Network address tung@caf.mit.edu

1 Introduction

A number of groups involved in the development or integration of process and device simulation programs have come to recognize the need for a profile interchange format [1], [2]. The intent of this document is to report ongoing work at MIT in the development of such a format, called here PIF. The philosophy, guidelines, and goals of the PIF have already been examined, including a discussion of the distinction between the format "syntax" and format "semantics". Here, we examine issues involved in the definition of a standard profile interchange format at the level of the actual information to be represented by the format (the semantics). This discussion is not couched in terms of a final proposal; rather, suggestions on possible solutions to the problems involved are made.

The primary goals we have for the PIF description of an object are:

Clarity of expression: The primitives used should be treated as uniformly as possible.

Uniformity: Understanding the syntax and semantics should require learning only the philosophy behind the PIF rather than the application of a large number of special cases. There is a lot to be said for "cleanliness" in the format.

General Mechanisms: The PIF representation should provide general methods for performing primitive tasks; larger problems of representation can then be built out of these smaller, simpler capabilities (i.e., general definition and reference mechanisms rather than hard-coded or ad-hoc implicit references.

If these goals can be satisfied, the resulting format should be easy and natural to use, extend, and implement.

The syntax used in this discussion will be the ascii form of the interchange format (PIF), discussed previously. The goals outlined above lead us, then, into describing the general philosophy and mechanisms in the PIF rather than a complete, item by item specification of the format. The full specification, which is ultimately very important and necessary, should follow naturally from the philosophy underlying the PIF.

2 Geometry Specification

The geometry specification mechanisms of the PIF allow for both a top-down and bottom-up hierarchical expression of geometry. This section describes both the basic geometry construct and the different types of geometry specifications.

2.1 Canonical Geometry Constructs

Specification of geometry information in the format is, of course, hierarchical in nature. In our format, the **structure** is composed of any number of **regions**, themselves composed of **boundary** and **grid**, and so on. In each geometry keyword, the same canonical construction is used:

(*geometry*
    [(**name** optional-name)]
    [(**reference** (**name** reference-name))]
    [(... defining geometry ...)]
    [(... transformations ...)]
    [(... attributes ...)]
)

More will be said later on the use of the **reference** keyword to access already defined geometries. The transformation and attribute constructs will be discussed later as well.

2.2 Geometry Types

There are two conceptually different types of geometry objects. First, there are those objects that have a distinct or absolute definition, such as lines, coordinates, faces, and solids. Secondly, there are also "structural objects" which are conceptually similar, but may be composed of different type of absolute geometry depending on the dimensionality of the problem. For instance, a "region" is a geometric object which is composed of a face in two dimensions, or a solid in three.

2.3 Primitive or Absolute Geometry

The primitive geometry objects are **coordinates**, **lines**, **faces**, and **solids**. Each is, perhaps implicitly, a three-dimensional object. For example, a **coordinate** has x, y, and z position information. **Solids** are defined by **faces**, which are defined by **lines**, which are finally defined by **coordinates**.

2.4 Structural Geometry

The conceptual objects out of which the structure is actually built include **nodes**, **boundaries**, **bounds**, **regions**, and *structures*. My gut feeling is that attribute information really should only be associated with the structural geometry of an object, and not with the absolute geometry. That is, a **coordinate** has no attributes associated with it, but a **node** does. Similarly, it makes sense to have attributes of a **boundary** or a part of a boundary (the **bound**), but not of a *line* itself. I think what this allows is for one to easily abstract out the true meaning of the data association; this meaning is maintained as the dimensionality of the problem changes.

For example, the two dimensional region defined as

```
(region
        (dimensions 2)
        (boundary
                (bound
                        (line ...)
                        (surface-states ...))
                (bound ...))
        (material
                (type silicon)))
```

can easily be abstracted up to three dimensions with only a change to the primitive geometry. No changes in the attribute information or hierarchy is required:

```
(region
        (dimensions 3)
        (boundary
                (bound
                        (face ...)
                        (surface-states ...))
                (bound ...))
        (material
                (type silicon)))
```

3 Definition - Reference Mechanism

This section gives a couple of examples of the definition and reference mechanisms in PIF. At any point that a geometry is needed, the geometry may be specified in either of two ways. First, the body of the geometry may actually be specified (leading to a very top-down approach). Alternatively, the body of the geometry may be specified by referring to an already defined geometry (leading to bottom-up definitions). For example:

```
(region
      (name region1)
      ...)
```

with a later reference:

```
(region
      (reference
            (name region1))
      (... transforms ...)
      ...)
```

3.1 Name Resolution Issues

There are several different approaches in the resolution of names in the above reference scheme. The basic problem is that the resolution of symbolic links should be as transparent to the user as possible. In some sense, a better expression of the above example (with the full type information included) might be:

```
(region
      (name %definition region1)
      ...)
```

with a later reference:

```
(region
      (name %reference region1)
      (... transforms ...)
      ...)
```

In this example, the resolution of the symbolic reference could be completely handled by the calling access function. That is, the definition reference mechanism may well rate a new intrinsic data type (on the level of %string or %real).

A second issue involves the resolution of the name itself. I am assuming that names are scoped in some sense. Resolution of the name "region1" within the set of region names requires looking back for the last definition of that name in the file.

4 Attribute Association

Attributes can be associated with any structural geometry. The general mechanism for association is to include the attribute construct within the definition of the geometry. Note that the attribute information may itself be complex.

```
(region
        ...
        (material
                (type silicon)
                (orientation 100))
        ...)
```

Note that the keyword is the type of the attribute itself. Those applications that do not recognize the keyword can skip the definition of the attribute entirely. The data is defined to apply over the entirety of the geometry defined. In the above example, the material properties apply to the whole region. Data applying to, say, a bound of the region would be associated within the **bound** construct.

4.1 Data Definition and References

Just as with geometry constructs, the same naming (definition) and reference mechanisms apply:

```
(material
        (name default-silicon)
        (type silicon)
        (orientation 100))
```

with a later reference of

```
(region
        ...
        (material
                (reference
                        (name default-silicon)))
        ...)
```

5 Tranformation of Coordinate Data

One of the primary goals of the hierarchical geometry specification is that higher level geometries be specified using lower lever geometries. It is very desirable that lower level geometry be "reusable" within other geometries. There are two types of needs in building up a large geometry. The first is to simply substitute pieces of similar dimensionality (tranform a region at one location into a region in another location). A second need is more involved: the ability to *change* the dimensionality of an object. For instance, a two-dimensional region may be *constructed* out of several one-dimensional regions. The two very different mechanisms involved will be discussed below.

5.1 Default Coordinate System

As mentioned before, each of the primitive geometry components is implicity a three-dimensional object. The assumed coordinate system is x into the silicon from the surface (in keeping with one-dimensional simulator convention), for y to be along the surface of the silicon, and for $z = x \times y$ (coming out of the paper). Unfortunately, the conventions used by one and two-dimensional simulators is in conflict. An alternative may be to define a default interpretation in two and three-dimensions which is different than that in one, but such a special case does not seem very clean.

5.2 Transformation Mechanisms

Each structural geometry construct may have a **transform** associated with it. This changes the values of the three-dimensional coordinates within the scope of that geometry. For instance, a coordinate defined as (1.0, 0.0, 0.0) may have a translation tranform at a higher level in the geometry, so that at that higher level the coordinate appears as (1.0, 1.0, 0.0). the general form of this translation construct would be:

```
(transform
      [(translate dx [dy])]
      [(rotate theta)]
      [(Tmatrix r11 r12 r21 r22 t1 t2)]
)
```

These transformations are shown for the two-dimensional case. A general approach to transformations is to use homogeneous coordinate systems, as described below.

5.2.1 Homogeneous Coordinates

A very clean and powerful mechanism for specification of geometric tranformations result from the use of homogeneous coordinate systems [3]. These allow translation, scaling, and rotations of objects (or, equivalently, coordinate systems) to be uniformly treated as matrix multiplication operations. A vector coordinate in two dimensions $(x\ y)$ becomes $(x\ y\ 1)$. The matrix operation for translation is $T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ dx & dy & 1 \end{bmatrix}$ while rotation is expressed with $R(\theta) = \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$. The real advantage of homogeneous coordinate tranformations is that successive transformations are very easily composited. A single tranformation may involve first a rotation, then a translation: $(x'\ \ y'\ \ 1) = (x\ \ y\ \ 1)R(\theta)T$. The cumulative tranformation matrix M, however, can be constructed once and used instead: $M = R(\theta)T$, so that $(x'\ \ y'\ \ 1) = (x\ \ y\ \ 1)M$.

Thus, an arbitray number of nested transformations may be applied (which is important in that we want to allow an arbitrary nesting of geometry specifications). Note also that the special form of homogeneous matrices allow efficient multiplication that is much less involved than the full matrix multiplication.

5.3 Construction Mechanisms

Unlike similar-dimensional *transformation* mechanisms, additional *construction* mechanisms are needed when a geometry is to be "constructed" from lower-dimensional objects. The constructors described in [2] are excellent examples of the need and utility of such a capability.

6 Additional Syntax Issues

While the syntax of the interchange format has been discussed previously [4], additional comments are appropriate to explain some of the syntactic choices made in the PIF as it has been used in this document.

6.1 Restricted - Data on Leaf Nodes

First of all, we have chosen to restrict actual data to the leaf nodes of the representation. Rather than use a highly position dependent format, we demand that every piece of information carrying any semantic weight be accompanied by a keyword defining the type of information involved. Note that this is a departure for the syntax of EDIF. We believe that this restriction increased readability and explicitness in the format, and allows for true extensibility in the format.

As an example, consider the two similar definitions below. The first is an EDIF-like construct, the second, a PIF equivalent:

```
(line 1 8 4 3 short 1.1 1. 1.2 .9 1.2 1.)
```

```
(line
        (Identifier 1)
        (Origin 8)
        (Terminus 4)
        (NumberInteriorPoints 3)
        (point 1.1 1.)
        (point 1.2 .9)
        (point 1.2 1.))
```

Of course, the primary disadvantage of the second specification is that it is much more lengthy than the first. The advantages, on the other hand, are that each of the implicitly defined (by position only) values are made explicit in the second. The addition of a parameter to the line statement at some future time, then, will not change the form of the **line** statement itself. A new attribute of the line statement would be added, and upward compatibility is trivially preserved.

The disadvantage is the increased length of the description. The resolution of this is accomplished through the keyword definition mechanism of EDIF, which has additional advantages as well.

6.2 Indentation notation

An alternative way of expressing the above *data structuring* is to include an explicit numbering for the indentation. This, for instance, is more in keeping with the format that the

SNC interactive browser uses to convey the data structuring to the user. For the above case, the same information could be presented as:

```
1. line
     2. Identifier %integer
     2. Origin %integer
     2. Terminus %integer
     2. NumberInteriorPoints %integer
     2. point %realarray
     2. point %realarray
     2. point %realarray
```

It should also be noted that several options are available in the SNC2PIF translation utility currently in use. These allow the suppression of type information, of all data, or of all array data. These options, then, generate an ascii "summary" rather than a complete PIF description from the local SNC database.

6.3 Keyword Definition

The keyword definition mechanism satisfies several very important needs. First, it provides a means for dramatically shortening the ascii representations of any construct in the PIF file. Secondly, it provides a very clean mechanism for the extension of the format, (both the extension of the "standard" format, and for "local" enhancements by the user). Finally, it provides a means for making the actual semantic conventions explicit.

In essence, the keyword definition construct allows the definition of a new keyword, and can make explicit the association of position dependent parameters with fully paranthesized substitutions. The concept is best explained with an example:

```
(keyword
     (name line)
     (formal id)
     (formal startid)
     (formal endid)
     (formal numpoints)
     (extra datapoints)
          (build
               (key lineobj)
                    (build
                         (key Identifier)
                         (type %string))
                         (value id))
                    (build
                         (key Origin)
          (type %string))
          (value startid))
     (build
          (key Terminus)
          (type %string))
          (value endid))...)
```

Note that the base-level data type is also specified in this construction. That is, it is made explicit that the **Origin** keyword will be a string or an integer, if that is its type. Once defined, the **line** construction as it appears in the first example above can be used inside the PIF description. To external programs referencing the description, they can directly access the subfields within the (line ...) construct through their full keywords (you might, for instance, request "line.Origin" from inside the application program).

6.3.1 Standard Keywords

The PIF "standard" consists of the agreed-upon keyword definitions. These can be included explicitly in a transmitted PIF file as a preamble of keyword definitions. As the "standard" evolves, then, these old descriptions will still be readable, since they are not coded into the *syntax* explicitly. The preamble provides a link between a fully EDIF-like definition of the PIF, and the more general SNC-like description of the PIF.

Having described keywords and the transformations between the short and full descriptions of a PIF construct, we can go on and presume a set of standard keywords and use the abbreviated form in the examples to follow.

7 Standardization Issues

The majority of this paper has discussed the appearance and organization of the ascii profile interchange format. Little has been said about the binary format. A couple of guidelines about the characteristics of the binary format follow.

7.1 Access Functions

While it is agreed that standardization of the interchange, ascii level PIF is critical, less attention has been paid to the need for standardization of interfaces between the tools and the local databases. Not only should it be possible to easily transfer wafer information between different sites, but it should also be relatively easy to transfer actual tools between different sites. Agreement upon what the interface between the local database and the tools themselves is thus also very important.

7.2 Conceptual Congruence

It is important that the "view" that applications tools have of the local and interchange formats be essentially identical. In fact, it should be possible to substitute the actual PIF representation with the local database entirely. This may require a different implementation of the access functions, but the application program should not care about how the information is actually stored.

8 Examples

Two examples are included here. The first is a simple, one-dimensional Suprem-III like geometry. The second is a PIF description for the information currently output by MINI-MOS. Note that these examples have not yet been made to conform to the description of PIF in this document. They only presented here as a quick starting point.

8.1 One-dimensional Example

```
(structure
     (header
          (tool Suprem-III)
          (date "November 9, 1986")
          (description "Example of 1D Structure PIF Description"))
     (dimensions 1)
     (units microns)
     (region
          (name "oxidation barrier")
          (boundary
               (point (coordinate 0.0))
               (point (coordinate 0.2)))
          (material
               (type nitride)))
     (region
          (name "pad oxide")
          (material
               (type oxide))
          (boundary
               (point (coordinate 0.2))   ;The origin actually lies "outside" the region here.
               (point (coordinate 0.3)))
          (tabular-profile
               (grid
                    (type difference)
                    (number-spaces 10)
                    (xvalues .01 .01 ... .01)
                    (layer-thickness 0.1)       ;These parameters specific to Suprem-III grids.
                    (layer-dx .01)
                    (layer-xdx .01))
               (boron
                    (concentration
                         (nodal-values 1.1e15 ...)))))
     (region
            (name "underlying silicon")
            (material
                (type silicon)
                (orientation 100))
            (transform                          ;In this example, the silicon grid starts at 0.0.
                (translate 0.3))    ;In the structure, then, all x pts have 0.3 added to them.
            (boundary
                (point (coordinate 0.0))
                (point (coordinate 1.0)))
            (analytic-profile
                (boron
                    (concentration
                        (constant-value 1e15)))
                (arsenic
                    (concentration
                        (gaussian
                            (prefactor 1e19)
                            (range 0.2)
                            (straggle 0.017)))
                    (implanted)))       ;Another example of a simulator-specific parameter.
            (thickness 1.0))
        (temperature                                                ;Also Suprem-III specific.
            (degreesC 1000.0)))
```

8.2 MINIMOS Example

```
(Contacts
    (Source
        (Voltage ...)
        (Current ...)
        (Hot-Electron-Current ...))
    (Gate
        (Voltage ...))
    (Drain
        (Voltage ...)
        (1D-Current ...)
        (2D-Current ...)
        (Avalanche-Current ...)
        (Hot-Electron-Current ...))
    (Bulk
        (Voltage ...)
        (Avalanche-Current ...)
```

```
            (Hot-Current ...)))
  (Physical-Parameters
        (Fermi-Voltage ...)
        (Flatband-Voltage ...)
        (Work-Function ...)
        (Terminal-Voltage ...)
        (Oxide-Capacitance ...)
        (Junction-Depth ...)
        (Subdiffusion ...)
        (Debye-Length ...)
        (Intrinsic-Concentration ...))
  (All-Regions
        (Nodal-Data
              (Rectangular-Grid
                    (X ...)
                    (Y ...))
              (Electrostatic-Potential ...))
        (Lateral-Offset-Data
              (Rectangular-Grid
                    (X ...)
                    (Y ...))
              (Lateral-Field ...)
              (Vertical-Offset-Data ...)
              (Rectangular-Grid
                    (X ...)
                    (Y ...))
              (Transverse-Field ...)))
  (Silicon-Substrate
        (Nodal-Data
              (Rectangular-Grid
                    (X ...)
                    (Y ...))
              (Doping-Concentration ...)
              (Electron-Fermi-Level ...)
              (Hole-Fermi-Level ...)
              (Electron-Concentration ...)
              (Hole-Concentration ...)
              (Avalanche-Generation ...)
              (Space-Charge ...)
              (Lateral-electron-mobility ...)
              (Transverse-electron-mobility ...)
              (Lateral-hole-mobility ...)
```

```
          (Transverse-hole-mobility ...)
          (Electron-temperature ...)
          (Hole-temperature ...))
     (Lateral-Offset-Data
          (Rectangular-Grid
               (X ...)
               (Y ...))
          (Lateral-electron-current ...)
          (Lateral-hole-current ...))
     (Vertical-Offset-Data
          (Rectangular-Grid
               (X ...)
               (Y ...))
          (Transverse-electron-current ...)
          (Transverse-hole-current ...)))
```

References

[1] A. R. Neureuther, "Profile Interchange Format," 1985. Working notes.

[2] S. Duvall and D. Lucey, "An Interchange Format for Process and Device Simulation," Jan. 1986. Personal communication.

[3] J. D. Foley and A. Van Dam, *Fundamentals of Interactive Computer Graphics*. Reading, Massachusetts: Addison-Wesley, 1984.

[4] D. S. Boning and T. Tung, "A proposal for a profile interchange format," June 1986. MIT CAF Project working notes.

END

3-87

DTIC